# CPROG Rapport för Programmeringsprojektet

Gruppnummer: 16

Gruppmedlemmar: Hanna Birgestam 000116-0068, Jessica Borch 920621-4109

*Skriv en kortfattad instruktion för hur programmeringsprojektet skall byggas och testas, vilka krav som måste vara uppfyllda, sökvägar till resursfiler(bildfiler/ljudfiler/typsnitt mm), samt vad en spelare förväntas göra i spelet, hur figurernas rörelser kontrolleras, mm.*

*Om avsteg gjorts från kraven på Filstruktur, så måste också detta motiveras och beskrivas i rapporten.*

*Fyll i 'check-listan', så att du visar att du tagit hänsyn till respektive krav, skriv också en kort kommentar om på vilket sätt du/gruppen anser att kravet tillgodosetts, och/eller var i koden kravet uppfylls.*

*Den ifyllda Rapportmallen lämnas in tillsammans med Programmeringsprojektet*. Spara rapporten som en PDF med namnet CPROG\_RAPPORT\_GRUPP\_NR.pdf (där NR är gruppnumret).

## Beskrivning

Spelet utspelar sig i rymden där spelaren är ett svart hål och ska suga åt sig olika fallande objekt mot sig. När spelaren kolliderar med ett fallande objekt ska det alltså försvinna. För varje objekt som spelaren samlar så ökas poängen. Om spelaren missar att fånga ett objekt så är spelet över.

## Instruktion för att bygga och testa

För att kunna bygga och testa spelet krävs det att SDL-biblioteken är installerade och tillgängliga i en subfolder vid namn ”lib” då denna sökväg till biblioteken är specificerad i vår Makefile. Kompilatorn ska även ha stöd för g++ och c++17. Vi har gjort ett avsteg från kraven på Filstruktur, vilket är att vår mapp för include-filer heter ”include” istället för ”includes”.

## Krav på den Generella Delen(Spelmotorn)

* 1. [ Ja/Nej/Delvis ] Programmet kodas i C++ och grafikbiblioteket SDL2 används.  
     **Kommentar**: Ja, spelet är kodat i C++ och vi har implementerat samtliga SDL2-bibliotek som efterfrågades
  2. [ Ja/Nej/Delvis ] Objektorienterad programmering används, dvs. programmet är uppdelat i klasser och använder av oo-tekniker som inkapsling, arv och polymorfism.  
     **Kommentar**: Vi har använt oss av upprepat arv med Element.h som superklass till GameElement.h och UIElement.h varpå de är superklass till andra klasser som är då uppdelade till rörliga och icke-rörliga. Vi valde just GameElement för rörliga objekt och UIElement för icke-rörliga objekt, däremot kunde de fått andra namn som DynamicElement och StaticElement för att göra det mer tydligt.
  3. [ Ja/Nej/Delvis ] Tillämpningsprogrammeraren skyddas mot att använda värdesemantik för objekt av polymorfa klasser.  
     **Kommentar**: Genom att i Element överskugga copy-konstruktor och tilldelningsoperator med delete och därmed förbjuda att dessa används i subklasserna. Vi säkerställer också att subklasserna endast kan instansieras dynamiskt genom att sätta konstruktorn privat(för lövklasserna, hos superklasserna är konstruktorn protected) och använder en publik funktion ”getInstance()” som använder sig av konstruktorn och returnerar ett pekarobjekt.
  4. [ Ja/Nej/Delvis ] Det finns en gemensam basklass för alla figurer(rörliga objekt), och denna basklass är förberedd för att vara en rotklass i en klasshierarki.  
     **Kommentar**: Ja, som förklarat i 3.2 så har vi de två klasserna GameElement och UIElement som agerar som basklass för rörliga och icke-rörliga objekt. Till UIElement har vi skapat två subklasser ”Button” och ”Label” som i sin tur har subklasser, medan för GameElement har vi ”Target” och ”Player” som inte har några.
  5. [ Ja/Nej/Delvis ] Inkapsling: datamedlemmar är privata, om inte ange skäl.  
     **Kommentar**: Alla datamedlemmar är privata och kan hämtas och ändras om de så bör med get- och set-funktioner.
  6. [ Ja/Nej/Delvis ] Det finns inte något minnesläckage, dvs. jag har testat och sett till att dynamiskt allokerat minne städas bort.  
     **Kommentar**: Genom att i Element ha en virtual destruktor så ser vi till att allt tas bort i hela ledet av arvshierarkin. Detta har vi också testat genom utskrifter i repsektive destruktor och sett att dessa anropas under programmets gång.
  7. [ Ja/Nej/Delvis ] Spelmotorn kan ta emot input (tangentbordshändelser, mushändelser) och reagera på dem enligt tillämpningsprogrammets önskemål, eller vidarebefordra dem till tillämpningens objekt.  
     **Kommentar**: Vi har implementerat i spelmotorn att det kan finnas både tangentbordshändelser och mushändelser. I Element har vi virtuella funktioner för båda typer av inputs så att dessa går att överskugga, dock har vi valt i vårt spel att överskugga mushändelser i UIElement och tangentbordhändelser i GameElement.
  8. [ Ja/Nej/Delvis ] Spelmotorn har stöd för kollisionsdetektering: dvs. det går att kolla om en Sprite har kolliderat med en annan Sprite.  
     **Kommentar:** För kollisionsdetektering anropas SDLs egna funktion SDL\_HasIntersection som ska vara True om två objekt överlappar. När det stämmer så anropas objektens egna funktioner vid kollision. Vi har alltså valt att kollision utifrån objektets rektangel och inte pixlar.
  9. [ Ja/Nej/Delvis ] Programmet är kompilerbart och körbart på en dator under både Mac, Linux och MS Windows (alltså inga plattformspecifika konstruktioner) med SDL 2 och SDL2\_ttf, SDL2\_image och SDL2\_mixer.  
     **Kommentar**: Ja, ingen plattformsspecifik kod har använts i vårt projekt.

## Krav på den Specifika Delen(Spelet som använder sig av Spelmotorn)

* 1. [ Ja/Nej/Delvis ] Spelet simulerar en värld som innehåller olika typer av visuella objekt. Objekten har olika beteenden och rör sig i världen och agerar på olika sätt när de möter andra objekt.  
     **Kommentar**: För själva spelet så har vi ett svart hål som agerar som protagonist och använder sig av Player-klassen, och ska fånga olika typer av objekt som faller mot den och använder sig av Target-klassen. Targets går inte att styra utan faller på sina egna villkor, medan Player går att styra med användargenererade händelser. När en Target kolliderar ska den försvinna från skärmen, och när Player kolliderar så initierar den en ny Target.
  2. [ Ja/Nej/Delvis ] Det finns minst två olika typer av objekt, och det finns flera instanser av minst ett av dessa objekt.  
     **Kommentar**: Ja, det finns en instans av Player och flera instanser av Target.
  3. [ Ja/Nej/Delvis ] Figurerna kan röra sig över skärmen.   
     **Kommentar**: Ja, Target förflyttar sig på y-axeln och Player och förflyttar sig på x-axeln.
  4. [ Ja/Nej/Delvis ] Världen (spelplanen) är tillräckligt stor för att den som spelar skall uppleva att figurerna förflyttar sig i världen.  
     **Kommentar**: Ja, varje steg som objekten tar syns på spelplanen.
  5. [ Ja/Nej/Delvis ] En spelare kan styra en figur, med tangentbordet eller med musen.  
     **Kommentar**: Ja, vi har valt att använda tangentbordshändelser, vänster- och högerpil, för att förflytta sin spelare.
  6. [ Ja/Nej/Delvis ] Det händer olika saker när objekten möter varandra, de påverkar varandra på något sätt.  
     **Kommentar**: När en Target möter en Player, minskar Target sin storlek och försvinner ur spelskärmen. För Player så tar den bort Target och genererar en ny Target.